```
【学習要項】
□ Phong shading
☐ Specular reflection
☐Bump map
□Normal mapping
□ Dummy texture
☐Bounding box
 【演習手順】
1. ピクセルシェーダー(static_mesh_ps.hlsl)で鏡面反射(スペキュラー)を表現する(フォンシェーディングに変更する)
         ①static_mesh クラスのコンストラクタ引数を.\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources\\Presources
         ②framework クラスの定数バッファ構造体(scene_constants)メンバ変数にカメラ位置を追加する
                   1: struct scene_constants
                   2: {
                   3:
                                  DirectX::XMFLOAT4X4 view_projection;
                   4:
                                 DirectX::XMFLOAT4 light_direction;
                  *5:
                                  DirectX::XMFLOAT4 camera_position;
                   6: };
         ③framework クラスの render メンバ関数でカメラ位置をセットしピクセルシェーダにバインドする
                   1: scene_constants data{};
                   2: DirectX::XMStoreFloat4x4(&data.view_projection, V * P);
                   3: data.light_direction = light_direction;
                  *4: data.camera_position = camera_position;
                   5: immediate_context->UpdateSubresource(constant_buffers[0].Get(), 0, 0, &data, 0, 0);
                   6: immediate_context->VSSetConstantBuffers(1, 1, constant_buffers[0].GetAddressOf());
                  *7: immediate_context->PSSetConstantBuffers(1, 1, constant_buffers[0].GetAddressOf());
         (5)HLSL ヘッダファイル(static_mesh.hlsli)のコードを変更する
                     1: struct VS_OUT
                     2: {
                                    float4 position : SV_POSITION;
                     3:
                     4:
                                   float4 color : COLOR;
                     5:
                                    float2 texcoord : TEXCOORD;
                  * 6:
                                    float4 world_position : POSITION;
                  * 7:
                                    float4 world_normal : NORMAL;
                     8: };
                     9: cbuffer OBJECT_CONSTANT_BUFFER : register(b0)
                   10: {
                   11:
                                    row_major float4x4 world;
                                    float4 material_color;
                   12:
                   13: };
                   14: cbuffer SCENE_CONSTANT_BUFFER : register(b1)
                   15: {
                   16:
                                    row_major float4x4 view_projection;
                   17:
                                    float4 light_direction;
                  *18:
                                    float4 camera_position;
                   19: };
         ⑥頂点シェーダー(static_mesh_vs.hlsl)のコードを変更する
                   1: VS_OUT main(float4 position : POSITION, float4 normal : NORMAL, float2 texcoord : TEXCOORD)
                   2: {
                   3:
                                  VS OUT vout;
                   4:
                                  vout.position = mul(position, mul(world, view_projection));
                   5:
                   6:
                                  vout.world_position = mul(position, world);
                   7:
                                  normal.w = 0;
                                 vout.world_normal = normalize(mul(normal, world));
                   8:
                   9:
                                  vout.color = material_color;
                  10:
```

11:

vout.texcoord = texcoord;

```
12:
        13:
                return vout;
        14: }
   (7)ピクセルシェーダー(static_mesh_ps.hlsl)のコードを変更する
        ※static_mesh クラスの render メンバ関数でコンスタントバッファオブジェクトをピクセルシェーダにもバインドする
         1: #include "static_mesh.hlsli"
         2:
         3: Texture2D color_map : register(t0);
         4: SamplerState point_sampler_state : register(s0);
         5: SamplerState linear_sampler_state : register(s1);
         6: SamplerState anisotropic_sampler_state : register(s2);
         7:
         8: float4 main(VS_OUT pin) : SV_TARGET
         9: {
        10:
              float4 color = color_map.Sample(anisotropic_sampler_state, pin.texcoord);
        11:
              float alpha = color.a;
        12:
              float3 N = normalize(pin.world_normal.xyz);
        13:
              float3 L = normalize(-light_direction.xyz);
        14:
              float3 diffuse = color.rgb * max(0, dot(N, L));
        15:
        16:
        17:
              float3 V = normalize(camera_position.xyz - pin.world_position.xyz);
        18:
              float3 specular = pow(max(0, dot(N, normalize(V + L))), 128);
        19:
        20:
              return float4(diffuse + specular, alpha) * pin.color;
        21: }
   ⑧ビルド・実行して鏡面反射光を確認する
        ※(7)18:行目の pow 関数の第2引数の値(128)を任意の値に変更して描画結果を確認する
2. バンプマップを扱えるようにするためにマテリアル情報(materials)を拡張する
   (1)static_mesh クラスのマテリアル情報構造体を変更する
         1: struct material
         2: {
         3:
                std::wstring name;
               DirectX::XMFLOAT4 Ka{ 0.2f, 0.2f, 0.2f, 1.0f };
         4:
               DirectX::XMFLOAT4 Kd{ 0.8f, 0.8f, 0.8f, 1.0f };
         5:
               DirectX::XMFLOAT4 Ks{ 1.0f, 1.0f, 1.0f, 1.0f };
         6:
        *7:
                std::wstring texture_filenames[2];
        *8:
                Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> shader_resource_views[2];
         9: };
   ② static_mesh クラスコンストラクタの MTL ファイルパーサー部でキーワード map_bump または bump を処理する
          1: if (0 == wcscmp(command, L"map_Kd"))
         2: {
          3:
                fin.ignore();
                wchar_t map_Kd[256];
          4:
                fin >> map_Kd;
          5:
          6:
          7:
                 std::filesystem::path path(obj_filename);
         8:
                 path.replace_filename(std::filesystem::path(map_Kd).filename());
        * 9:
                 //materials.rbegin()->texture_filename = path;
        *10:
                 materials.rbegin()->texture filenames[0] = path;
                 fin.ignore(1024, L'\u00e4n');
         11:
         12: }
        *13:
            else if (0 == wcscmp(command, L"map_bump") || 0 == wcscmp(command, L"bump"))
        *14: {
        *15:
                 fin.ignore();
        *16:
                 wchar_t map_bump[256];
                 fin >> map_bump;
        *17:
        *18:
```

```
std::filesystem::path path(obj_filename);
        *19:
                path.replace_filename(std::filesystem::path(map_bump).filename());
        *20:
                materials.rbegin()->texture_filenames[1] = path;
        *21:
        *22:
                fin.ignore(1024, L'\u00e4n');
        *23: }
   ③ static_mesh クラスコンストラクタでテクスチャをロードする
   ④ static_mesh クラスの render メンバ関数でカラーマップとバンプマップをピクセルシェーダーにセットする
       immediate_context->PSSetShaderResources(0, 1, material.shader_resource_views[0].GetAddressOf());
       immediate_context->PSSetShaderResources(1, 1, material.shader_resource_views[1].GetAddressOf());
   (5)法線マッピングの処理をピクセルシェーダ(static_mesh_ps.hlsl)に実装する
         1: #include "static_mesh.hlsli"
         2:
         3: Texture2D color_map : register(t0);
        *4: Texture2D normal_map : register(t1);
         5: SamplerState point_sampler_state : register(s0);
         6: SamplerState linear_sampler_state : register(s1);
         7: SamplerState anisotropic_sampler_state : register(s2);
         8:
         9: float4 main(VS_OUT pin) : SV_TARGET
        10: {
        11:
              float4 color = color_map.Sample(anisotropic_sampler_state, pin.texcoord);
        12:
              float alpha = color.a;
              float3 N = normalize(pin.world_normal.xyz);
        13:
        14:
        *15:
              float3 T = float3(1.0001, 0, 0);
        *16:
              float3 B = normalize(cross(N, T));
        *17:
              T = normalize(cross(B, N));
        18:
        *19:
              float4 normal = normal_map.Sample(linear_sampler_state, pin.texcoord);
        *20:
              normal = (normal * 2.0) - 1.0;
        *21:
              normal.w = 0;
              N = normalize((normal.x * T) + (normal.y * B) + (normal.z * N));
        *22:
        23:
        24:
              float3 L = normalize(-light_direction.xyz);
        25:
              float3 diffuse = color.rgb * max(0, dot(N, L));
        26:
        27:
              float3 V = normalize(camera_position.xyz - pin.world_position.xyz);
        28:
              float3 specular = pow(max(0, dot(N, normalize(V + L))), 128);
        29:
        30:
              return float4(diffuse + specular, alpha) * pin.color;
        31: }
   ⑥ビルド・実行して法線マッピング表現を確認する
       ※(5)11:行目および 19:行目のサンプラーステートを変更した場合の描画結果の違いを確認する
3. MTL ファイルを持たない OBJ ファイル(cube.obj, torus.obj)をロード・描画できるように改造する(下記問題を解決する)
   (1) MTL ファイルのオープンに失敗した場合はアサーションによりプログラムが停止する(コメントアウトする)
   (2)マテリアル情報(materials)のサイズがゼロになるので描画が出来ない(ダミーマテリアルの追加)
       1: if (materials.size() == 0)
       2: {
            for (const subset& subset : subsets)
       4:
       5:
              materials.push_back({ subset.usemtl });
       6:
       7: }
   (3)テクスチャが存在しないのでピクセルシェーダーで正しい動作が行えない(ダミーテクスチャ作成関数の実装)
```

1: HRESULT make_dummy_texture(ID3D11Device* device, ID3D11ShaderResourceView** shader_resource_view,

```
DWORD value/*0xAABBGGRR*/, UINT dimension)
         2:
         3: {
              HRESULT hr{ S_OK };
         4:
         5:
         6:
              D3D11 TEXTURE2D DESC texture2d desc{};
              texture2d_desc.Width = dimension;
              texture2d_desc.Height = dimension;
         8:
         9:
              texture2d_desc.MipLevels = 1;
        10:
              texture2d_desc.ArraySize = 1;
        11:
              texture2d_desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
        12:
              texture2d_desc.SampleDesc.Count = 1;
              texture2d_desc.SampleDesc.Quality = 0;
        13:
              texture2d_desc.Usage = D3D11_USAGE_DEFAULT;
        14:
              texture2d_desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
        15:
        16:
        17:
              size_t texels = dimension * dimension;
        18:
              unique_ptr<DWORD[]> sysmem{ make_unique< DWORD[]>(texels) };
        19:
              for (size_t i = 0; i < texels; ++i) sysmem[i] = value;</pre>
        20:
              D3D11_SUBRESOURCE_DATA subresource_data{};
        21:
        22:
              subresource_data.pSysMem = sysmem.get();
        23:
              subresource_data.SysMemPitch = sizeof(DWORD) * dimension;
        24:
        25:
              ComPtr<ID3D11Texture2D> texture2d;
        26:
              hr = device->CreateTexture2D(&texture2d_desc, &subresource_data, &texture2d);
        27:
              _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
        28:
        29:
              D3D11_SHADER_RESOURCE_VIEW_DESC shader_resource_view_desc{};
        30:
              shader_resource_view_desc.Format = texture2d_desc.Format;
              shader_resource_view_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
        31:
              shader_resource_view_desc.Texture2D.MipLevels = 1;
        32:
        33:
              hr = device->CreateShaderResourceView(texture2d.Get(), &shader_resource_view_desc,
        34:
                shader_resource_view);
        35:
              _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
        36:
        37:
              return hr;
        38: }
    (4)ダミーカラーマップテクスチャの作成例
        make_dummy_texture(device, material.shader_resource_views[0].GetAddressOf(), 0xFFFFFFFF, 16);
    (5)ダミー法線マップテクスチャの作成例
        make_dummy_texture(device, material.shader_resource_views[1].GetAddressOf(), 0xFFFF7FFF, 16);
4. 衝突判定等で活用するためにバウンディングボックス(境界ボックス)情報を static mesh クラスのメンバ変数に追加する
    ①メッシュをちょうど囲うのに必要な大きさの四角い箱
    ②メッシュ頂点座標(各 xyz 座標)の最大値と最小値で表現できる
    ③static_mesh クラスのコンストラクタで計算する
    ④ワイヤフレームのボックスを描画することで static_mesh オブジェクトの境界ボックスを可視化する
    ⑤static_mesh オブジェクト描画時にピクセルシェーダーを変更できる仕組みを考える
5. スタティックメッシュを使った 3 D ゲームを制作する
【評価項目】
□スタティックメッシュクラスの改良
```

□3Dゲーム制作